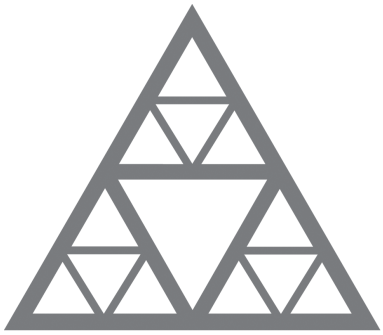
École Nationale des Ponts et Chaussées



École des Ponts

ParisTech

Techniques et développement logiciel

Plateforme d'investissement et pricer d'options

Florian Arfeuille, Gaspard Beaudouin, Yentl Collin, Clément Dureuil

Septembre 2023 - Janvier 2024

Introduction et objectifs

Contents

Introduction et objectifs

Notre projet initial de technique de développement de logiciel consistait à développer un pricer d'options pour les marchés financiers. Nous avons choisi de nous concentrer sur deux types d'options : les options européennes et les options américaines. Une option européenne est un instrument financier dérivé conférant à son détenteur le droit, mais non l'obligation, d'acheter (option d'achat) ou de vendre (option de vente) un actif sous-jacent à un prix prédéterminé, appelé prix d'exercice, à une date d'échéance spécifiée. La caractéristique distinctive d'une option européenne est que l'exercice ne peut avoir lieu qu'à la date d'échéance, contrairement à l'option américaine qui permet l'exercice à n'importe quel moment avant ou à la date d'échéance.

Cependant, au cours de l'implémentation de ces méthodes, nous avons constaté qu'elles étaient peu nombreuses et ne contribuaient pas suffisamment à notre projet. C'est à ce moment que nous avons eu l'idée d'une plateforme d'investissement, permettant aux utilisateurs de gérer leur portefeuille en achetant et vendant des actions au prix du marché. Bien que cette plateforme soit fictive en raison des licences financières nécessaires pour acheter des actions et des collaborations avec des instituts financiers, nous avons réussi à obtenir le vrai prix des actions et donc être au plus proche de la réalité. Nous avons maintenu la fonctionnalité d'achat d'options développée précédemment.

Bien que des plateformes similaires existent déjà, cette expérience nous a permis de mieux comprendre le fonctionnement de ces systèmes et d'apporter notre propre perspective éducative au projet.

Nous nous sommes donc lancés dans ce projet, et nous vous raconterons les différentes difficultés rencontrées dans chaque sous-partie correspondante.

1 Pricer

1.1 Flask

Nous avons fait un site à l'aide de Flask, un micro framework open-source de développement web en Python. Les fonctions de route peuvent recevoir des requêtes HTTP (GET, POST, etc.) et renvoyer des réponses HTTP. Flask facilite la gestion des données de requête et de réponse. Dans une application Flask, les différentes parties du site sont gérées par des "routes". Une route est associée à une URL spécifique et est gérée par une fonction particulière.

Initialement, nous avons envisagé l'utilisation d'un template prédéfini, tel que CookieCutter, pour notre projet. Cependant, nous avons finalement opté pour une approche plus

minimaliste, en démarrant presque de zéro. Cette décision a été influencée par un template de base qui nous avait été fourni lors d'un hackathon en début de projet.

Notre structure comprenait un fichier main.py dédié à la gestion des routes, un fichier style.css pour styliser notre page web, un index.html servant de page d'accueil, et un fichier JavaScript pour la gestion des boutons et des événements. Cette configuration de départ simple nous a permis de construire notre application de manière plus flexible et personnalisée, en nous adaptant précisément aux besoins spécifiques de notre projet.

Ici, la route '/' est associée à la page d'acceuil du pricer, où l'utilisateur peut choisir le type d'option qu'il désire acheter avec deux boutons.

Nous créons donc des fichiers european.html, american.html, resultat_european.html et resultat_american.html. Nous mettons donc un bouton 'calcul' sur european.html et american.html qui redigirent vers resultat. Des forms dans european.html et american.html permettent de récuperer les données de l'options que l'utilisateur souhaite acheté : prix actuel, maturité ... (cf 1.2).

Nous avions alors d'abord mis deux boutons sur cette page, qui redirigaient vers resultat_american.html ou resultat_european.html .

Figure 1: Page d'accueil et du pricer d'option americaine.

Pourtant, nous avons ensuite réalisé qu'il était plus facile avec flask (et sans js) d'afficher le resultat, sans ouvrir une nouvelle page html resultat_american ou resultat_european de cette façon, avec une nouvelle route '/resultat', la méthode 'POST' et avec la La structure conditionnelle `{% if result %} ... {% endif %}` pour conditionner l'affichage de contenu en fonction de la présence ou de la valeur d'une variable.

Ensuite, pour optimiser notre application, nous avons cherché des solutions permettant d'afficher des graphiques et des résultats sans recharger entièrement la page. Cette approche répondait à deux impératifs majeurs. Premièrement, l'utilisation d'une API pour la génération de graphiques était limitée à 25 requêtes quotidiennes comme nous le détaillerons dans la partie suivante. Il était donc essentiel de minimiser les appels à cette API afin de préserver notre quota journalier, ce qui rendait le rechargement complet de la page pour chaque affichage de résultat non viable. Deuxièmement, pour améliorer l'expérience utilisateur, il est préférable d'éviter le rechargement constant de la page, surtout en cas de

connexion Internet faible.

Pour surmonter ces contraintes, nous avons adopté une stratégie combinant l'usage de méthodes GET et l'implémentation de JavaScript. Cette approche nous a par exemple permis d'améliorer l'expérience utilisateur sur les pages de marché d'options. Grâce à cette technique, nous avons pu afficher le prix d'une option choisie par l'utilisateur - prix basé sur le choix d'échéance et de prix d'exercice choisis par l'utilisateur - de manière fluide et instantanée, sans nécessiter le rechargement complet de la page. Concernant l'affichage des graphiques, nous avons opté pour l'utilisation en plus d'un modal, un choix dont l'intérêt et le mode d'emploi seront expliqués plus en détail dans la section 1.4.

Un des principaux défis que nous avons rencontrés concernait la gestion de Git, qui s'est avérée complexe, particulièrement au début de notre projet. Étant donné que chaque membre de l'équipe était assigné à une tâche spécifique sur le site, nous travaillions sur des branches distinctes. Cependant, cette méthode de travail entraînait fréquemment des modifications simultanées sur les mêmes segments de code, ce qui générait des conflits. La fusion des branches, ou "merge", s'est souvent révélée être une tâche délicate à naviguer. Cette situation nécessitait une attention particulière et une coordination accrue pour assurer une intégration harmonieuse des différents éléments de code.

1.2 API

L'un des défis majeurs de notre projet consistait à obtenir en temps réel les prix des actions. Après des recherches approfondies, nous avons rapidement opté pour l'utilisation d'une API. Celle-ci agit comme un intermédiaire, permettant à deux applications distinctes de partager des informations ou de collaborer sans exposer leur code interne complet. Les développeurs utilisent les API pour intégrer des fonctionnalités spécifiques, accéder à des données ou interagir avec des services tiers dans leurs propres applications.

Dans notre cas, nous utilisons une API pour récupérer les données des prix des actions depuis des sites internet. Après avoir exploré différentes options, Yahoo Finance semblait être un choix évident. Cependant, lors de nos tests, nous avons rapidement constaté qu'obtenir les données était difficile en raison du manque de documentation.

En poursuivant nos recherches, nous avons découvert Financial Modeling Prep, qui autorise les interactions avec des programmes externes. De plus, une version gratuite nous offre une grande quantité de données. Après avoir créé un compte et généré une clé API, nous pouvons, grâce à des commandes simples en Python, récupérer nos données en temps réel.

Pour des raisons esthétiques, nous souhaitions générer un graphique illustrant les variations du cours des actions au cours d'une journée. Après avoir vérifié les fonctionnalités de Financial Modeling Prep, nous avons constaté que ces données étaient disponibles moyennant un paiement. Ne souhaitant pas payer, nous avons recherché pendant un certain temps et trouvé une solution sur un nouveau site avec une API qui nous fournit ces données, Alpha Vantage. Sur ce site, nous pouvons récupérer les valeurs des actions à toutes les heures de la journée. Cependant, le défaut est qu'il est impossible d'obtenir les données du jour actuel.

Nous avons donc programmé notre graphique sur le prix de l'action de la journée précédente. Bien que ces données reflètent la journée précédente, nous avons tenté de camoufler cela en affichant le graphique des prix de l'action entre l'heure d'ouverture de la bourse et l'heure actuelle à laquelle le programme tourne. Nous simulerons ainsi le jour actuel avec les valeurs d'hier.

Cependant, nous avons dû faire face à quelques limitations. Nous sommes limités à 25 requêtes API par jour envers ce serveur, ce qui complexifie nos tests de programmes. De plus, nous avons constaté que notre API ne renvoie pas de données les weekends et après 18h lorsque la bourse est fermée. Pour pallier cela, nous avons décidé d'afficher les prix du vendredi si le programme est utilisé un lundi. Ainsi, veuillez noter que notre plateforme n'affichera ni les données ni les prix en dehors des heures de marché.

Nous avons également dû choisir un marché financier, car sur l'API, nous avions le choix de choisir n'importe quel marché financier (Américain, les marchés Européens n'étant pas présents sur l'API). Nous avons choisi le marché NASDAQ, l'un des plus importants aux États-Unis. Ses horaires d'ouverture sont du lundi au vendredi de 14h30 à 21h (heure de Paris). Nous avons opté pour ce marché car il est l'un des plus importants, et la plupart des actions des entreprises que nous connaissons sont cotées sur ce marché. Comme nous prenons le prix de l'action du jour précédent, nous pouvons adapter les horaires américains aux horaires de la bourse en France. Ne pas choisir un marché était impossible car certains symboles d'actions sont répétitifs entre les marchés. Un symbole du marché Nasdaq ne représente pas forcément la même action sur le New York Stock Exchange. Pour être sûr de l'action que l'on veut, nous avons choisi un marché financier précis.

1.3 Modele du pricer

Les options d'achat *(resp. de vente*) sont des produits dérivés d'actions. Il en existe de toute sortes. Nous nous focaliserons dans ce projet uniquement sur les options américaines et européennes. A la date de signature du contrat, l'acheteur de l'option fixe :
- l'actif sous jacent qui correspond dans le cas de notre site à l'action que l'on voudra acheter *(resp. vendre*) au terme du contrat.
- la date d'échéance T qui correspond à la durée du contrat.
- le prix d'exercice K qui correspond au prix auquel il pourra acheter l'actif sous-jacent à la date T. Si il décide d'acheter *(resp. de vendre)* le sous-jacent à la date T, alors on dit que celui detu-ient de le contrat **exerce l'option**. Dans la suite on se restreindra uniquement qu'aux option d'achat par soucis de clareté.

Dans le cas de l'option européenne, on ne peut exercer l'option uniquement à la date T, alors que dans le cas de l'option américaine, on a la possibilité d'exercer (c'est-à-dire acheter l'actif sous-jacent au prix K à n'importe quel moment avant la date T).

Comme, l'acheteur à le droit d'acheter ou non, il faut qu'il paye un certain montant pour bénéficier de ce droit. C'est l'objet de cette partie : il faut calculer le prix de ce contrat.

1.3.1 Volatilité

Pour calculer le prix de l'option, il faut tout d'abord definir la notion de **volatilité**. La volatilité d'une action mesure la variation des prix de celle-ci sur une période de temps donnée. Elle est souvent utilisée comme indicateur de risque. Plus la volatilité est élevée, plus les prix de l'action sont susceptibles de fluctuer de manière significative. On peut estimer la volatilité implicitement en utilisant le cours historique avec N obeservations du cours l'action à l'aide de cette formule:

$$\text{Volatilité Historique} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(u_i - \bar{u})^2}$$

- N est le nombre d'observations.
- $u_i = \ln\left(\frac{S_i}{S_{i-1}}\right)$.
- S_i est le cours de l'action à l'observation i.
- $\bar{r}$ est la moyenne des u_i.

1.3.2 Pricing d'une option européenne

On peut maintenant introduire la formule de Black-Scholes, qui permet de calculer le prix d'une option européenne sous certaines conditions que l'on va supposer vérifiées dans notre projet (taux de rendemment sans risque constant, pas de cout de transaction, ...).
La formule de Black-Scholes pour une option d'achat est la suivante :

$$C = S_0 N(d_1) - K e^{-rT} N(d_2) \tag{1}$$

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \qquad d_2 = d_1 - \sigma\sqrt{T}$$

Ainsi, l'utilisateur n'aura qu'à indiquer le nom de l'actif sous jacent, la date d'échéance T, et le prix d'exercice K. On se fixe un taux de rendement sans risque arbitraire $r = 0.02$. Ainsi grâce à l'API, on peut récupérer le prix de cloture du cours de l'action indiqué par l'uilisateur et caluler la volatilité implicite associée sur une durée d'un an. On peut de plus récupérer le prix S_0 de l'action au moment de l'achat du contrat grâce à une fonction qui fourni le prix en direct de l'action. On a donc tous les éléments pour pricer notre option européenne et fournir le prix du contrat à l'utilisateur sur notre plateforme.

Dans le cas d'une option d'achat, la formule est presque similaire et fait appel aux mêmes varbiables. **On ne se limitera uniquement aux options d'achats par manque de temps.**

1.3.3 Pricing d'une option américaine

Par soucis de place disponible dans le compte rendu, nous n'allons pas détailler la méthode de pricing d'une option américaine. En effet, elle est est un peu plus complexe et fait appel au modèle de Cox-Ross et à une simultation des cours des prix par une méthode de Monte-Carlo. Notons que la fonction **american_option_price** prend les mêmes paramètres en entrée que la fonction **european_option_price**.

1.4 Afficher le cours d'une action

Notre objectif était de permettre à l'utilisateur de sélectionner une action soit par son nom, soit par son symbole, pour ensuite afficher un graphique représentant l'évolution de sa valeur sur les 7 dernières heures. Pour ce faire, nous avons mis en place deux champs de saisie : un pour le nom de l'action et un autre pour son symbole. Ces champs étaient connectés aux données boursières via une API. Ensuite, nous avons développé une fonction nommée AfficherPopup, qui avait pour rôle de transférer les données de l'action vers un modal pour l'affichage du graphique.

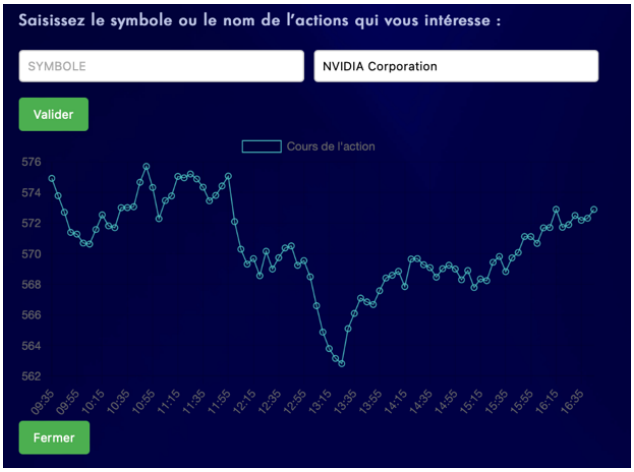


Figure 2: Modal affichant le graphique représentant l'action NVDA

L'emploi d'un modal pour cette tâche présentait plusieurs avantages significatifs. Les modals offrent une manière élégante de présenter des informations sans nécessiter une redirection vers une autre page, permettant ainsi à l'utilisateur de rester dans son contexte actuel et d'expérimenter une navigation plus fluide. De plus, la capacité d'ouvrir et de fermer le modal à la manière d'une pop-up donnait aux utilisateurs un meilleur contrôle sur la gestion de l'espace de la page. En termes de performance, l'utilisation de modals se révélait bénéfique puisque le contenu n'est chargé que lorsqu'il est nécessaire, réduisant ainsi le temps de chargement initial de la page et les ressources utilisées. Cette approche était d'autant plus cruciale compte tenu de notre limite de 25 requêtes API par jour, nous obligeant à minimiser

leur usage.

Au début de notre projet, pour l'affichage des graphiques, nous avions envisagé de développer une fonction qui récupérerait les données de l'API pour générer une image de graphique classique avec Matplotlib. Cependant, après avoir implémenté cette solution, nous avons réalisé qu'elle n'était ni esthétiquement satisfaisante ni particulièrement pratique pour permettre à l'utilisateur de visualiser efficacement l'évolution des valeurs boursières dans le temps.

Nous avons donc réévalué nos besoins et décidé de nous orienter vers une solution qui offrirait non seulement un aspect visuel plus attrayant, mais aussi une interaction plus dynamique avec les données. Dans cet esprit, nous avons choisi d'utiliser la bibliothèque Chart.js. Cette bibliothèque nous a permis de créer des graphiques esthétiques avec une visualisation des données interactives, offrant une expérience utilisateur améliorée et une compréhension plus intuitive des fluctuations des actions au fil du temps. L'intégration de cette bibliothèque se faisait simplement par l'importation du script chart.js.

Nous avons rencontré certains défis sur cette affichage, notamment en matière de gestion de la mémoire. Initialement, seul le premier graphique demandé s'affichait correctement, les tentatives suivantes échouant à afficher les données d'une nouvelle demande de l'utilisateur. Après plusieurs tests, nous avons compris qu'il était nécessaire de détruire le contenu du modal avant de le mettre à jour avec de nouvelles informations. Cette découverte, comme d'autres, ont été cruciale pour assurer le bon fonctionnement de notre fonctionnalité d'affichage de graphiques.

2 La plateforme d'investissement

2.1 Mise à jour du site : d'un Pricer à une plateforme d'investissement

Nous avons décidé d'étendre notre projet de TDLOG et de réaliser une plateforme d'investissement. Pour cela, nous avons dû mettre en place une base de données (explication sur cela en section suivante).

La route de base '/' redirige vers login.html si l'utilisateur n'est pas connecté. S'il n'a pas de compte, il peut en créer un avec un `<button>` qui redirige vers register.html. Il peut ensuite cliquer sur le `<button>` log in pour finalement se connecter, et arriver sur la page d'accueil.

L'utilisateur voit son solde et peux de rajouter de l'argent en inscrivant le montant souhaité dans le formulaire `<montant>`. Il peut aussi voir le contenu de son portfeuille dans la page d'acceuil. Dans l'onglet action, il a la possibilité de chercher une entreprise et de consulter son cours en bourse. Il peut aussi acheter ou vendre des actions. Il y a alors un formulaire pour choisir le nom de l'entreprise et le nombre d'actions que l'utilisateur souhaite acheter ou vendre.

De la même manière, il peut acheter et vendre des options américaine ou européennes dans les onglets correspondants. L'utilisateur doit renseigner le nombre d'options souhaités, le prix d'exercice et la date d'échéance. Il peut cliquer sur un boutton pour afficher le prix calculé par nos fonctions (prix actuel, calcul de volatilité,...) pour proposer un prix de l'option (que ce soit européenne ou américaine), afin de l'acheter ou la vendre.

2.2 Base de données utilisateurs

Avec SQL Alchemy, on définit le chemin de la base de données pour SQLAlchemy. On initialise l'extension SQLAlchemy avec l'application Flask, puis on configure l'extension Flask-Migrate avec l'application Flask et l'instance SQLAlchemy. On crée enfin une table pour notre base de données.

```
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.secret_key = bb 'clee secrete'
db = SQLAlchemy(app)
migrate = Migrate(app, db)
with app.app_context():
    db.create_all()
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)
    balance = db.Column(db.Float, default=500.0)
    stock_portfolio = db.Column(db.JSON, default={})
    european_option_portfolio = db.Column(db.JSON, default={})
    american_option_portfolio = db.Column(db.JSON, default={})
```

Listing 1: Code Python

Notre route de base '/' redirige vers '/login' si l'utiliseur existe et est connecté. Si l'utilisateur n'a pas de compte, l'utilisateur peut en céer un en cliquant sur un bouton qui redirige vers la route '/register'. On recupère username et password avec un `request.form`, à partir desquel on crée un User, et on l'ajoute a notre base de donnée.

Une fois l'utilisateur connecté, on accède facilement aux informations de l'utilisateur dans le html en donnant des paramètres à render_template dans le main :

```
return render_template('index.html', username=user.username, balance=user.
    balance, portfolio_value=portfolio_value, stock_portfolio=user.
    stock_portfolio, real_user=user)
```

Listing 2: Accéder aux informations de l'utilisateur sur le site, dans la route '/' ici

Le contenu des portefeuilles est le suivant :

- `american_option_portfolio['AAPL'][K=200][T=1]`=2 signifie que le user possède deux options d'achat américaines de 'AAPL', de prix d'exercice K=200, de maturité T= 1 an. Respectivement
- `european_option_portfolio['AAPL'][K=200][T=1]`= 2 pour une option européen.

Un des problèmes rencontré a été la mise à jour en temps réel de la base de donnée. Prenons l'example de la route '/buy_stocks'. Après l'achat d'une action il faut donc modifier le dictionnaire **user.stock_portfolio** et **user.balance** en conséquence. Avant de comprendre notre erreur, on modifiait directement **user.balance** et **user.stock_portfolio**.

```
stock_name = request.form.get('stock_name_buy') # Recuperer le nom
stock_price = prix_actuel(stock_name)  # et le prix actuel de l'action
if user.balance >= number_of_stocks * stock_price:
    user.balance -= number_of_stocks * stock_price
    print('user.balance after purchase:', user.balance)
    # Mettre a jour son portefeuille
    if stock_name in user.stock_portfolio:
        user.stock_portfolio[stock_name] += number_of_stocks
    else:
        user.stock_portfolio[stock_name] = number_of_stocks
    user.stock_portfolio = new_stock_portfolio
    db.session.commit()
```

Listing 3: Code pour l'achat d'actions

Le problème c'est que l'user.balance était mis à jour correctement sur le site mais pas l'user.stock_portfolio. Ce problème s'explique par la nature immuable de "user.stock_portfolio". Il faut donc créer une copie de user.stockportfolio. On peut donc faire des modifications suite à l'achat d'actions sur cette copie et à la fin faire user.stock_portfolio = copy_stock_portfolio.

2.3 Derniers ajouts

Lors de l'utilisation de notre pricer, il est impératif de saisir correctement l'orthographe de l'action que l'on souhaite traiter. À cette fin, nous avons implémenté des suggestions qui présentent les différents noms ou symboles avec le préfixe que vous avez saisi, des entreprises du Nasdaq (on se restreint à ce marché). Certains symboles peuvent être contre-intuitifs, comme par exemple AAPL pour Apple. Cette particularité a engendré plusieurs erreurs, car en cas de saisie incorrecte d'un symbole, l'API renvoie une valeur None. Cette situation entraînait deux problèmes majeurs. Premièrement, le bug ne se manifestait qu'au moment de l'affichage du graphique, ce qui rendait sa détection tardive. Deuxièmement, chaque tentative d'affichage, même infructueuse, était comptabilisée dans notre limite quotidienne de 25 requêtes API. Par conséquent, il était impératif de réduire au maximum les erreurs de saisie de l'utilisateur pour économiser nos requêtes. C'est pourquoi nous avons décidé d'implémenter

un système de suggestions. Cette fonctionnalité guide l'utilisateur lors de la saisie, réduisant ainsi le risque d'erreurs et, par extension, le nombre de requêtes API gaspillées. Cette approche avait pour but d'optimiser l'utilisation de nos ressources limitées tout en améliorant l'expérience utilisateur en fournissant une interface plus intuitive et moins sujette aux erreurs.

Pour informer l'utilisateur en cas de symbole ou de nom incorrect, nous avons ajouté l'affichage d'une fenêtre signalant que le nom ou le symbole n'est pas répertorié sur notre marché. Cette vérification s'effectue automatiquement lorsque l'utilisateur quitte la zone de saisie, et ainsi avant qu'il ne demande l'affichage du graphique, on ne perd donc pas une requête inutilement.

2.4 Peaufinement du style

Nous avons utilisé CSS pour cela. Nous avons mis en place une animation qui réalise un dégradé du noir sur notre image d'arrière-plan sur 6s. Il faut notamment utiliser :

```
.fade-in { ...
    position: relative;
    background: #000 url('nice_background.jpg') center/cover;}
.fade-in::before { ...
    opacity: 0;
    animation: fadeInBackground 6s ease-in-out forwards;}
@keyframes fadeInBackground {
    0%, 100% { opacity: 0.25; }
    100% { opacity: 1; } }
```

Listing 4: Animation de Fondu en CSS

De plus, nous avons créé des classes pour les différents boutons dans le fichier .css. L'un des problèmes principaux était que parfois des styles ne s'appliquaient pas sur toute la page, et donc des inputs avaient une présentation différente. Il semblait qu'il y avait des conflits dans le fichier CSS. En définissant des styles à l'intérieur du HTML, nous avons réussi à appliquer le style voulu sur toute la page.

Entre autres, nous avons ajouté des styles `button:hover` qui permettent de faire légèrement varier la couleur du bouton que l'utilisateur survole avec son curseur.

Nous avons changé la police avec `@font-face` pour la rendre plus moderne que la police de base.

Nous avons ajouté une classe tableau pour afficher le portfolio de l'utilisateur de manière claire, en accedant aux données de notre base de données.

Figure 3: Capture d'écran de la page "Actions"

3 Conclusion

Nous avons développé un site web qui a évolué d'une simple application de tarification d'options pour devenir une plateforme d'investissement. Nous sommes ravis d'avoir progressivement enrichi notre projet. L'utilisateur peut se connecter sur une interface où il a la possibilité d'ajouter des fonds à son compte, consulter les cours des actions en bourse, effectuer des transactions d'achat et de vente d'actions, ainsi que réaliser des opérations d'achat et de vente d'options.

D'un point de vue développement logiciel, nous avons pu découvrir et apprendre à utiliser Flask, HTML, CSS. Nous vu comment utiliser des API et une biliothèque d'affichage graphique sur HTML. Nous avons également appris à configurer une base de données avec SQLAlchemy.
Enfin, ce travail de groupe nous a permis de mieux utiliser Git et GitHub.

Il reste cependant de nombreux points à améliorer, tels que la possibilité de visualiser l'historique du portefeuille de l'utilisateur à travers un graphique illustrant l'évolution de son capital. Nous envisageons d'ajouter une barre de recherche avec un menu déroulant pour faciliter la navigation. De plus, l'intégration d'indicateurs visuels, tels que des flèches rouges ou vertes, pourrait rendre la plateforme plus attrayante en offrant une meilleure compréhension de la tendance des cours des actions. Nous envisageons également d'explorer une expérience "multijoueur" où les utilisateurs peuvent visualiser les portefeuilles d'autres participants et participer à un classement basé sur leur efficacité. Enfin, ce qu'il nous manque comme fonctionnalité serait de pouvoir exercer l'option d'achat et de pouvoir acheter des options de vente.